Assignment 1

Part I:

Answer 1

Source Code github repo link

How to run program

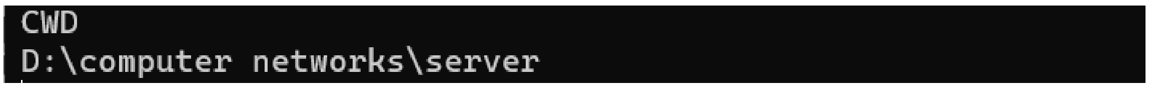
First run the server by following command

```
pyhton3 server.py
```

After running sever.py run client.py by following command

```
python3 client.py
```

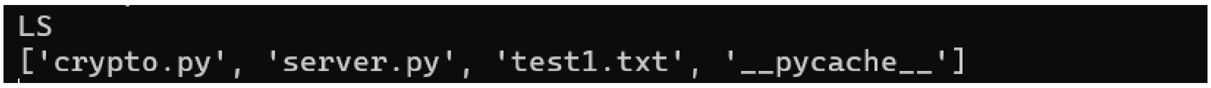
To know the current directory of the server command for that is given below.



```
CWD
D:\computer networks\server
```

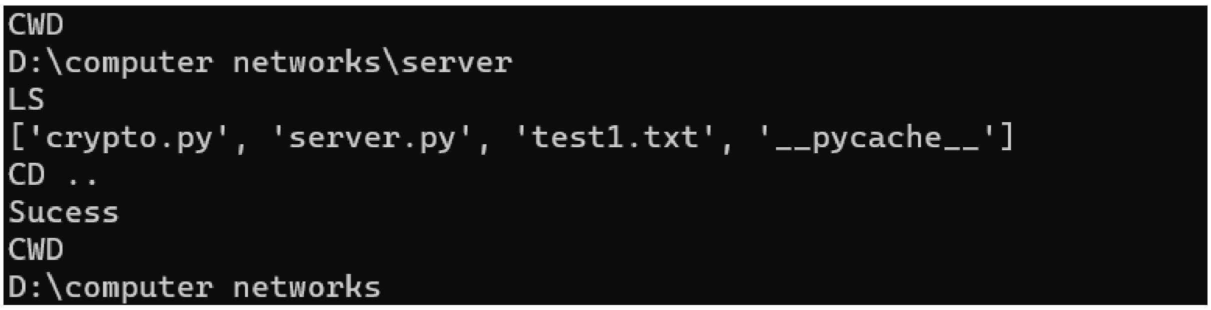
You can see the output in the next line to it.

To know the list of directories and files in the current writing directory give the below command.



```
LS
['crypto.py', 'server.py', 'test1.txt', '__pycache__']
```

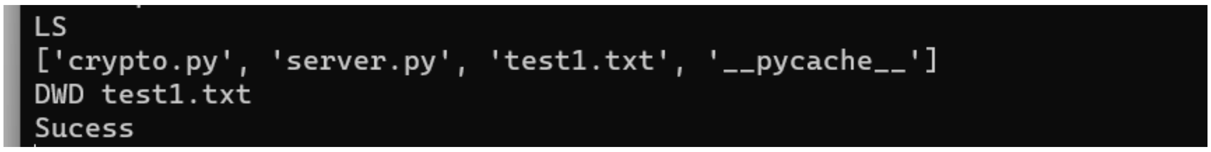
To change the current directory of the server you can give the below command if your filename is valid then you will get success status otherwise error.



```
CWD
D:\computer networks\server
LS
['crypto.py', 'server.py', 'test1.txt', '__pycache__']
CD ..
Sucess
CWD
D:\computer networks
```

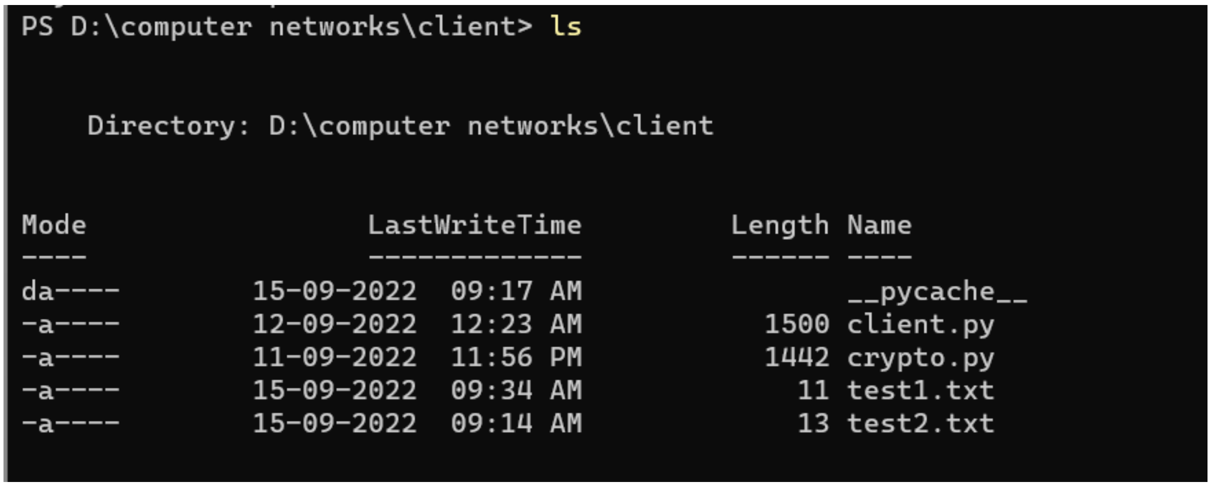
You can see the the CD command “ .. “ path means go to parent directory you can see that the given implementation is right by checking again current working directory.

Now we we will download “text1.txt” file.



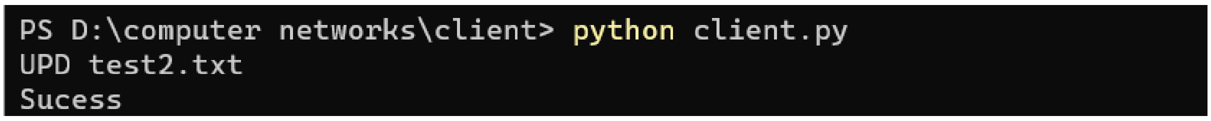
```
LS
['crypto.py', 'server.py', 'test1.txt', '__pycache__']
DWD test1.txt
Sucess
```

Now if you see the text1.txt file in your client working directory



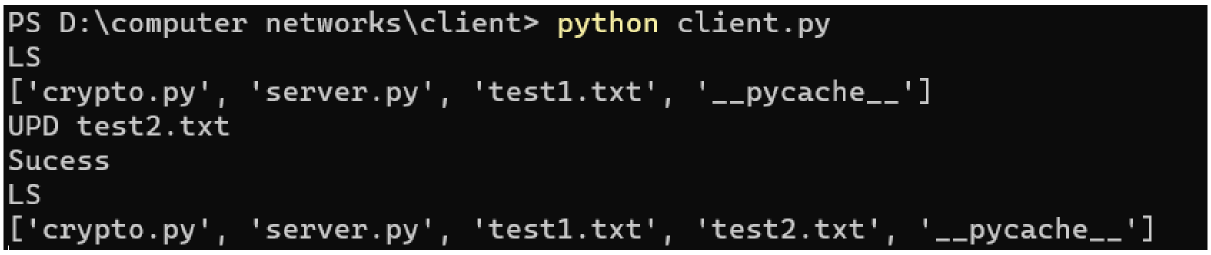
```
PS D:\computer networks\client> ls


    Directory: D:\computer networks\client


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
da----        15-09-2022  09:17 AM                __pycache__
-a----        12-09-2022  12:23 AM           1500 client.py
-a----        11-09-2022  11:56 PM           1442 crypto.py
-a----        15-09-2022  09:34 AM             11 test1.txt
-a----        15-09-2022  09:14 AM             13 test2.txt
```

Now we will upload text2.txt file. Note that it will be uploaded on current working directory of the server file.



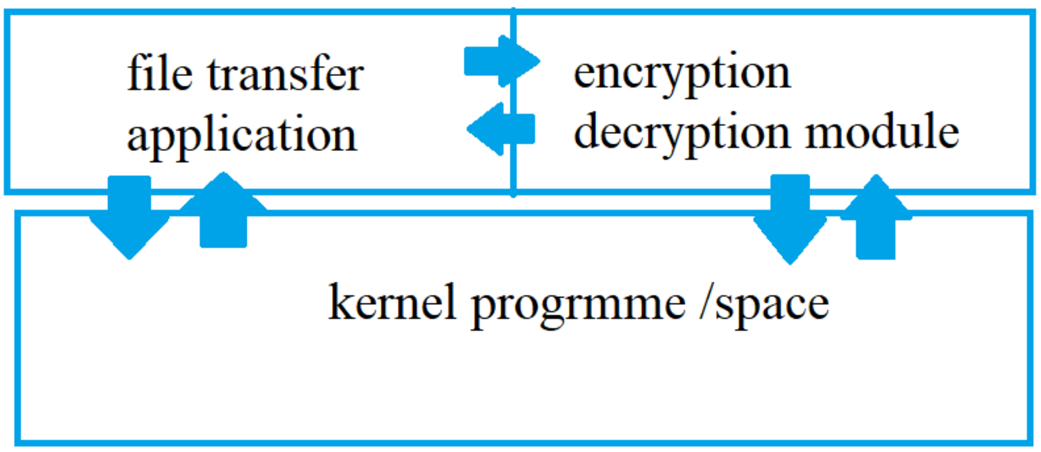
```
PS D:\computer networks\client> python client.py
UPD test2.txt
Sucess
```

If you check your server’s current working directory you will see the test2.txt file



```
PS D:\computer networks\client> python client.py
LS
['crypto.py', 'server.py', 'test1.txt', '__pycache__']
UPD test2.txt
Sucess
LS
['crypto.py', 'server.py', 'test1.txt', 'test2.txt', '__pycache__']
```

Implementation details and Testing



We have total 3 python files [server.py , client.py , crypto.py]. Server.py and client.py we will take it as file transfer application. File transfer call will call system calls and use functionality of crypto.py file. Crypto.py file calls system calls and some python API. Figure shows the same.

Server.py and client.py files are using TCP protocol for connection and file transfer. These file are using decode and encode functions defined in crypto.py. Port is 7778.

Protocol :

If client want to know about current working directory, different files then it needs to send request to server. Format of this request is pickle encoded. In pickle encoding you can encode list of strings in binary bits and decode it to again list of strings.

Example –

Pickle.dumps([“CD”, “<file/dir name>”]) -> 010111…

Pickle.loads(010111…) -> [“CD”, “<file/dir name>”]

when client is downloading file then server will send it in packets form its format is string encoded in binary. But before sending file first it will be encoded and after that strings will be encoded to binary number and then it will be transferred. Uploading format is also same. There is not any header at application layer.

There are three scheme for encryption and decryption part. First is plain text, next is ceaser cypher in that we will change charater to its next to next character in a cyclic way. And in the last scheme we will reverse words. Crypto.py will make first new file in which it will write encrypted data and then it will return that file in decryption part is also same and it will create file with same name and old file will be deleted after decryption.

Associated Challenges :

The biggest challenge was in upload and download file how to send whole file but it was done by sending some amount of data in 1 trip and again if there is more data then we will send data again.

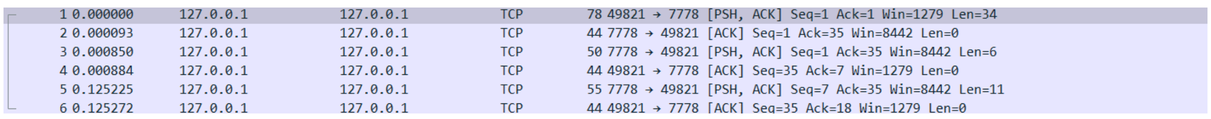
Other challenge was how to encrypt file it was done by duplicating and deleting files. To do that I have to use lots of function calls from os library.

It was great assignment I learnt lots of concept and problem solving to overcome these challenges.

Here JPG, or video files or any other format file is not supported but I think it can be implemented by using this project.

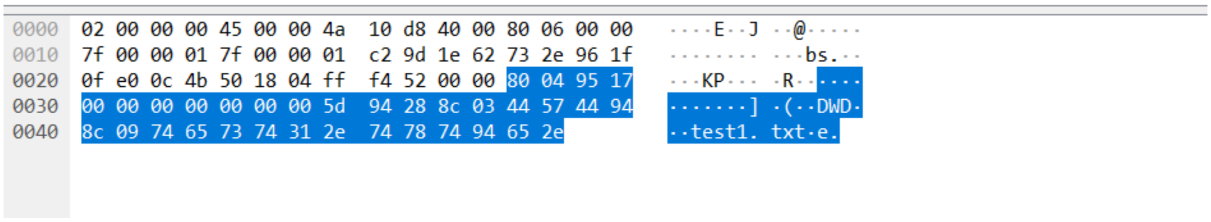
Test the data is encrypted correctly:

When we run wireshark and give commands which is for downloading file then we capture total 6 packets 3 of which is data exchange.



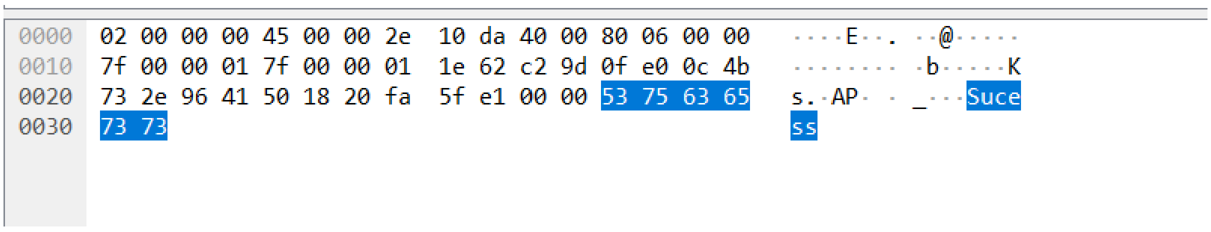
```
1 0.000000    127.0.0.1     127.0.0.1     TCP    78 49821 → 7778 [PSH, ACK] Seq=1 Ack=1 Win=1279 Len=34
2 0.000093    127.0.0.1     127.0.0.1     TCP    44 7778 → 49821 [ACK] Seq=1 Ack=35 Win=8442 Len=0
3 0.000850    127.0.0.1     127.0.0.1     TCP    50 7778 → 49821 [PSH, ACK] Seq=1 Ack=35 Win=8442 Len=6
4 0.000884    127.0.0.1     127.0.0.1     TCP    44 49821 → 7778 [ACK] Seq=35 Ack=7 Win=1279 Len=0
5 0.125225    127.0.0.1     127.0.0.1     TCP    55 7778 → 49821 [PSH, ACK] Seq=7 Ack=35 Win=8442 Len=11
6 0.125272    127.0.0.1     127.0.0.1     TCP    44 49821 → 7778 [ACK] Seq=35 Ack=18 Win=1279 Len=0
```

First instruction by client



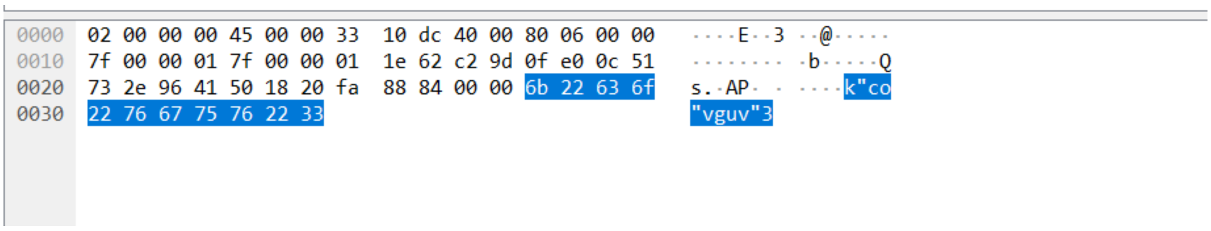
```
0000  02 00 00 00 45 00 00 4a  10 d8 40 00 80 06 00 00   ····E··J ··@·····
0010  7f 00 00 01 7f 00 00 01  c2 9d 1e 62 73 2e 96 1f   ········ ···bs.··
0020  0f e0 0c 4b 50 18 04 ff  f4 52 00 00 80 04 95 17   ···KP··· ·R······
0030  00 00 00 00 00 00 00 5d  94 28 8c 03 44 57 44 94   ·······] ·(··DWD·
0040  8c 09 74 65 73 74 31 2e  74 78 74 94 65 2e         ··test1. txt·e.
```

Response by server



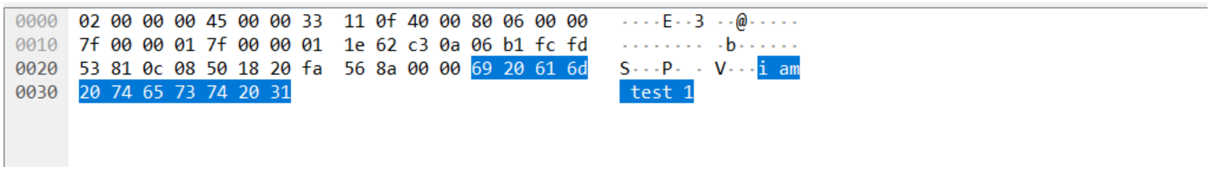
```
0000  02 00 00 00 45 00 00 2e  10 da 40 00 80 06 00 00   ····E··. ··@·····
0010  7f 00 00 01 7f 00 00 01  1e 62 c2 9d 0f e0 0c 4b   ········ ·b·····K
0020  73 2e 96 41 50 18 20 fa  5f e1 00 00 53 75 63 65   s.·AP· · _···Suce
0030  73 73                                              ss
```

Caeser encryption:



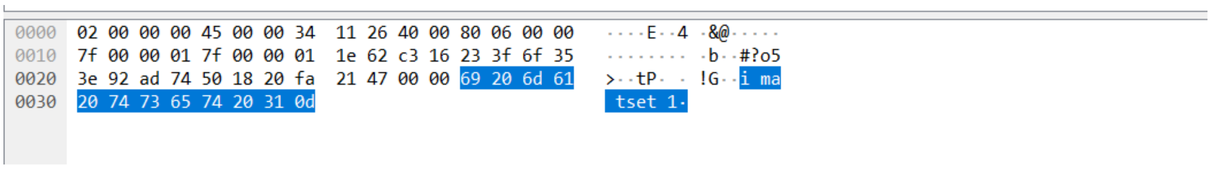
```
0000  02 00 00 00 45 00 00 33  10 dc 40 00 80 06 00 00   ····E··3 ··@·····
0010  7f 00 00 01 7f 00 00 01  1e 62 c2 9d 0f e0 0c 51   ········ ·b·····Q
0020  73 2e 96 41 50 18 20 fa  88 84 00 00 6b 22 63 6f   s.·AP· · ····k"co
0030  22 76 67 75 76 22 33                               "vguv"3
```

Plain text encryption :



```
0000  02 00 00 00 45 00 00 33  11 0f 40 00 80 06 00 00   ····E··3 ··@·····
0010  7f 00 00 01 7f 00 00 01  1e 62 c3 0a 06 b1 fc fd   ········ ·b······
0020  53 81 0c 08 50 18 20 fa  56 8a 00 00 69 20 61 6d   S···P· · V···i am
0030  20 74 65 73 74 20 31                                test 1
```

Reverse word encryption :



```
0000  02 00 00 00 45 00 00 34  11 26 40 00 80 06 00 00   ····E··4 ·&@·····
0010  7f 00 00 01 7f 00 00 01  1e 62 c3 16 23 3f 6f 35   ········ ·b··#?o5
0020  3e 92 ad 74 50 18 20 fa  21 47 00 00 69 20 6d 61   >··tP· · !G··i ma
0030  20 74 73 65 74 20 31 0d                             tset 1·
```

Part II:

Answer 2

a) Data in 1 packet = 100.1 kilobytes = 800.8 kilobits

$$\text{Transmission Delay} = \frac{number\ of\ bits\ to\ be\ transfered}{transmission\ rate}$$

Link 1 delay = 2.002 ms

Link 2 delay = 8.008 ms

Link 3 delay = 4.004 ms

Total = 14.014 ms

b) Data in 1 packet = (10000+100) bytes = 10.1 kilobytes = 80.8 kilobits

   Now there will be pipelining in packet transfer link 1 has higher transmission rate so, if we observe R1 then R1 will always have non empty queue of packets after it has received 1 packet. Delay for link 1 will be pipelined with Link 2 so, in total calculation of delay effective delay for link 1 will be 1 packet only. For R2 link 3 has higher transmission rate so, at R2 there will not be any packet queue so, for link 3 will also have effective delay of 1 packet. For link 2 there is queue at R1 so, it need to send all packets so, for it effective delay will be 10 packets.

   Formula total delay = 1*(Link1 delay) + 10*(Link2 delay)+ 1*(Link3 delay)
   Link 1 delay = 0.202 ms
   Link 2 delay = 0.808 ms
   Link 3 delay = 0.404 ms

   Total delay = 0.202 + 8.08 + 0.404 = 8.686 ms

c) Data in 1 packet = (2000+100) bytes = 16.8 kilobits

   Formula for the total delay will be similar
   Link 1 delay = 0.042 ms
   Link 2 delay = 0.168 ms
   Link 3 delay = 0.084 ms

   Total delay = 8.526 ms

d) Data in 1 packet = 8.8 kilobits
   Link 1 delay = 0.022 ms
   Link 2 delay = 0.088 ms
   Link 3 delay = 0.044 ms

   Total delay = 8.866 ms

Lowest delivery time is when there is 50 packets so, we can choose to make 50 packets from full data.

Answer 3 :

Propogation delay per bit = $\frac{length\ of\ the\ link}{propogation\ speed}$

Propogation delay per bit = $\frac{10000}{2 * 10^8}$ = $0.5 \times 10^{-4}\ s$

Maximum bit R1 can send before R2 receive a bit = $Propogation\ delay\ per\ bit \times transmission\ rate$ = $0.5 \times 10^7\ bits$

Bit width = $\frac{lenght\ of\ the\ link}{number\ of\ bits\ in\ link}$ = $\frac{10000}{0.5 \times 10^7}$ = 2×10^{-3} meters

Answer 4:

When we download files we have generally download speed around 1MBps so here we will take it as 1MBps as transmission speed.

a) In non persistent connection with each object we will have 1 more RTT. Because connection will be closed after server sent 1 object.
Page load time =$\sum (2 \times RTT + object\ size/$transmission speed)
Page load time = (20+1)+ 10*(20+100) = 1221 ms

b) In persistent connection every object will be send over same connection so there will be no need for RTT for each object.
Page load time = $RTT + \sum(RTT + objectsize/transmissionspeed)$
Page load time = 10 + (10+1) + 10*(10+100) = 1121 ms

c) In persistent + pipelined request when data is being send by server at that time request is also being send by client so, 2 RTT for connection and 1 RTT for first web request.
Page load time = 30 + (1)+10*(100) = 1031 ms

Part III

Answer 5

TFTP :

Application layer protocol. Trivial File transfer protocol is stock version of FTP. The port number of it is 69. RFC code is 1350.

LPD:

Line printer daemon was designed for printer sharing. It is application layer protocol it the part that receives and processes the request. Port number is 515. RFC 1179.

SNMP:

Simple network management protocol. It is also application layer protocol. It is a way that servers can share information about their current state, and also a channel through which administration can change the variables. Port number is 161 and 162. RFC 1157

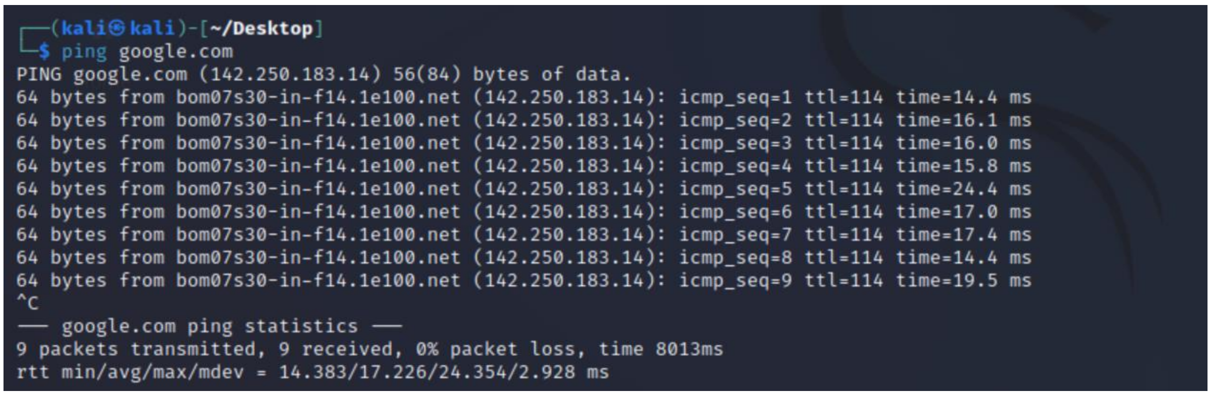
DHCP :

Dynamic host configuration Protocol. It is application layer protocol. It gives IP addresses to hosts. Port number is 67, 68. RFC code is RFC 2131.

X windows:

It defines protocol for the writing of graphical user interface- based client/sever application. Its port number is 6000 and increases by 1 for each server. RFC code 1013

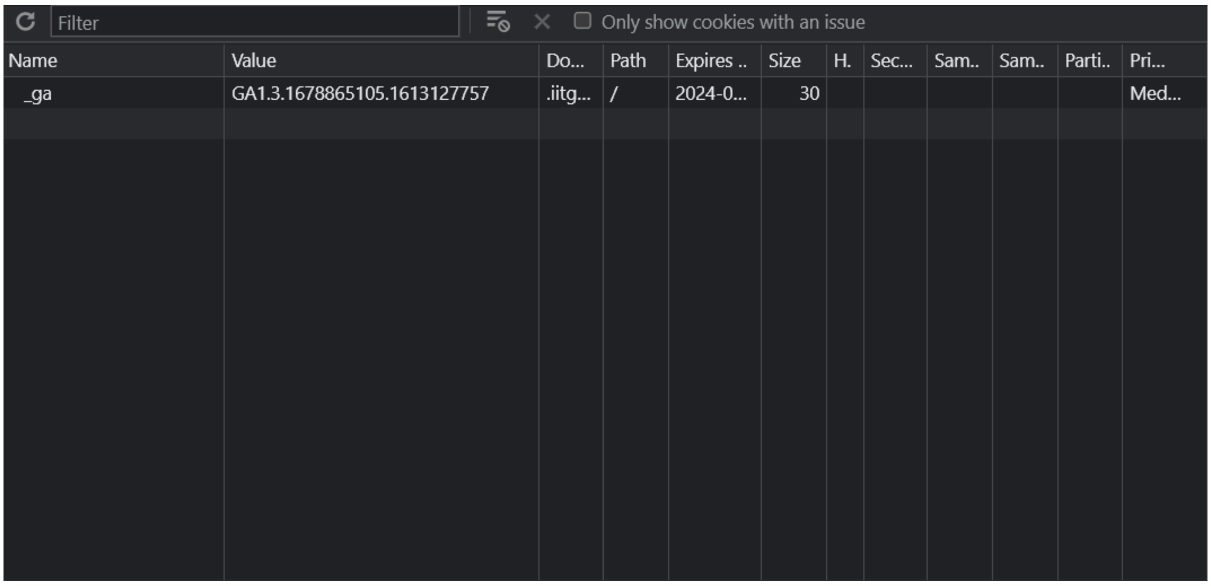
To calculate RTT(round trip time) of any network we can run ping and host name in kali shell after some packet transfer we can press Crtl+C and you will see the rtt time at the last line it is in ms. You can see below.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ ping google.com
PING google.com (142.250.183.14) 56(84) bytes of data.
64 bytes from bom07s30-in-f14.1e100.net (142.250.183.14): icmp_seq=1 ttl=114 time=14.4 ms
64 bytes from bom07s30-in-f14.1e100.net (142.250.183.14): icmp_seq=2 ttl=114 time=16.1 ms
64 bytes from bom07s30-in-f14.1e100.net (142.250.183.14): icmp_seq=3 ttl=114 time=16.0 ms
64 bytes from bom07s30-in-f14.1e100.net (142.250.183.14): icmp_seq=4 ttl=114 time=15.8 ms
64 bytes from bom07s30-in-f14.1e100.net (142.250.183.14): icmp_seq=5 ttl=114 time=24.4 ms
64 bytes from bom07s30-in-f14.1e100.net (142.250.183.14): icmp_seq=6 ttl=114 time=17.0 ms
64 bytes from bom07s30-in-f14.1e100.net (142.250.183.14): icmp_seq=7 ttl=114 time=17.4 ms
64 bytes from bom07s30-in-f14.1e100.net (142.250.183.14): icmp_seq=8 ttl=114 time=14.4 ms
64 bytes from bom07s30-in-f14.1e100.net (142.250.183.14): icmp_seq=9 ttl=114 time=19.5 ms
^C
--- google.com ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8013ms
rtt min/avg/max/mdev = 14.383/17.226/24.354/2.928 ms
```

Cookies

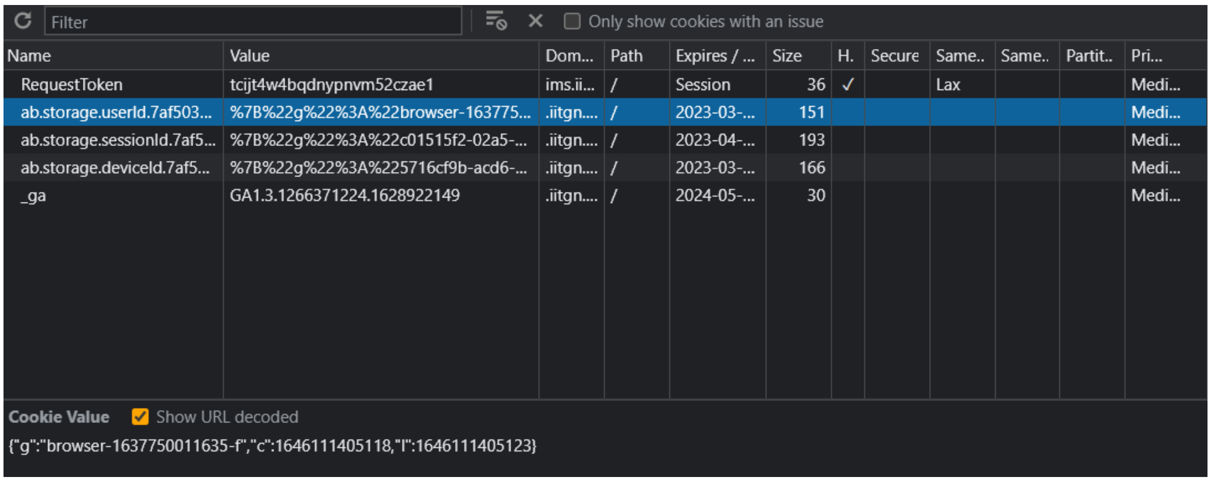
Before login



Filter | Only show cookies with an issue

Name	Value	Do...	Path	Expires ..	Size	H.	Sec...	Sam..	Sam..	Parti..	Pri...
_ga	GA1.3.1678865105.1613127757	.iitg...	/	2024-0...	30						Med...

After login



Filter | Only show cookies with an issue

Name	Value	Dom...	Path	Expires / ...	Size	H.	Secure	Same..	Same..	Partit..	Pri...
RequestToken	tcijt4w4bqdnypnvm52czae1	ims.ii...	/	Session	36	✓		Lax			Medi...
ab.storage.userId.7af503...	%7B%22g%22%3A%22browser-163775...	.iitgn....	/	2023-03-...	151						Medi...
ab.storage.sessionId.7af5...	%7B%22g%22%3A%22c01515f2-02a5-...	.iitgn....	/	2023-04-...	193						Medi...
ab.storage.deviceId.7af5...	%7B%22g%22%3A%225716cf9b-acd6-...	.iitgn....	/	2023-03-...	166						Medi...
_ga	GA1.3.1266371224.1628922149	.iitgn....	/	2024-05-...	30						Medi...

Cookie Value ✓ Show URL decoded

{"g":"browser-1637750011635-f","c":1646111405118,"l":1646111405123}

After login I am seeing 4 new cookies. In all 4 domain is iitgn.ac.in and there is session detail. It means that this key will be invalid after that time. Sizes of all the cookies are different. If we check Show URL decoded then we can see all the cookies are dictionary type which has key and value in all the cookies dictionary sizes are 4.